


CMR-520 Collection aggregated information for granules

This is an ops concept for  [CMR-520](#) - JIRA project doesn't exist or you don't have permission to view it. .

Patrick and other client developers want to have information per collection on things like:

- Do granule in this collection have cloud cover specified?
 - This would be used to change the GUI to indicate if they can filter on that field.
- What additional attributes are specified by the granules in the collection and what are the values that are used?
 - For example we would say granules in AST_L1A have the additional attribute "Band10_Available" and they specify values "Yes, band is acquired" and "No, band was not acquired". Other collections use the values "ON" and "OFF".

As Patrick stated in his comment this is different than facets but it's similar. We want to know which values are used within a field across all granules.

Implementation Options

Maintain a mapping of unique values to counts per collection for granules

During indexing we would increment or decrement the counts of values for a particular field in a collection. Storage of counts per field could be kept in Elasticsearch or as a table in Oracle.

Benefits of this approach

- Very fast
 - It would be very fast to return the aggregated values. All of the work is done during indexing.
- Puts no load on elasticsearch.
 - Because the aggregation is done during indexing.

Challenges

- More work relative to the other solution.
- Concurrent processing of counts is difficult.
 - Parallel processing of a counts that can increase and decrease is difficult. Imagine two processes that are concurrently handling granules in the same collection. If the two processes both process a granule at the same time that impact a count they will both fetch the original count increment it and save it. One count will overwrite the other count effectively losing one count. There are various ways to handle this. One is transactions in Oracle. This would potentially cause performance issues. Another is use of version in Elasticsearch. If we attempt to update a document containing counts in Elasticsearch and in the meantime another process already updated it we could redo the count manipulations with the latest version of the document. There are also ways of structuring the counts using CRDTs that always allow concurrent updates to merge without losing information.
- Does not handle numeric fields or string fields with many distinct values.
 - Counts really only work for string fields without many distinct values. If there are too many unique values we'd have to store all of them to make this particular solution work.

Elasticsearch Aggregations

Aggregations could be used to collection unique values for string fields and find min and max values for numeric fields. The following example query from marvel shows an aggregation search across granules in one collection. It finds the minimum and maximum cloud cover values along with additional attribute aggregations. The additional attributes are aggregated within each unique add. attrib. This allows us to get the top 10 unique string values and min and max float or integer values. This is only a subset of the final aggregations we would need to collect. We'd need to add the other nested additional attribute fields and any other granule fields needed by clients.

Requesting aggregated values takes a long time even on a single collection. The example search below takes about 14 seconds in workload. The time spent is in the aggregations. The time take seems to be mostly linear with the addition of each new aggregation to calculate. In order to provide information in a timely manner we would need to periodically collect this information and cache it. Since this is a relatively expensive query we wouldn't want to compute it separately on each app server. We would probably run this as a clustered job (on a single instance) and store the computed value in a central location (Oracle table or elastic) to provide access to all the app servers.

```

GET /1_c14758250_lpdaac_ecs/granule/_search
{
  "query": { "filtered": {
    "query": { "match_all": {} },
    "filter": {
      "term": {
        "collection-concept-id": "C14758250-LPDAAC_ECS"
      }
    }
  } },
  "size": 0,
  "aggs": {
    "min_cloud_cover": {
      "min": {
        "field": "cloud-cover"
      }
    },
    "max_cloud_cover": {
      "max": {
        "field": "cloud-cover"
      }
    },
    "additional_attributes": {
      "nested": {
        "path": "attributes"
      },
      "aggs": {
        "names": {
          "terms": {
            "field": "attributes.name",
            "size": 30
          },
          "aggs": {
            "strings": {
              "terms": {
                "field": "attributes.string-value",
                "size": 10
              }
            },
            "min-floats": {
              "min": {
                "field": "attributes.float-value"
              }
            },
            "max-floats": {
              "max": {
                "field": "attributes.float-value"
              }
            },
            "min-ints": {
              "min": {
                "field": "attributes.int-value"
              }
            },
            "max-ints": {
              "max": {
                "field": "attributes.ints-value"
              }
            }
          }
        }
      }
    }
  }
}

```

Benefits of this approach

- Relatively easy to implement.
- Client responses will be very fast and cheap to perform

Downsides and Challenges

- Cached information is out of date.
 - Since we would collect this information with a job and cache it it will be out of date.
 - Given that the information is an aggregation of terms, mins, and maxes being slightly out of date is not an issue.
- Puts unknown additional workload on the Elasticsearch cluster
 - We don't currently know how this impacts performance of concurrent searches or cached data. We need to generate a full aggregation query and run it during a workload run across all collections to determine what the impact is to search performance.

API Design

We would add a new endpoint in the search application `/granule_values_by_collection` that would allow retrieval of aggregated granule values by collection. Numeric fields would be returned with minimum and maximum values of all granules within that collection. String fields would be returned with the top N values. Counts would not be included in these results. The value of N is TBD. The fields which are included in this is TBD as well. Nested fields like additional attributes would have values grouped by additional attribute name.

The API would only support JSON initially. A concept-id parameter indicating the collection concept id to fetch would be required.

Example

```
curl -XGET
https://cmr.earthdata.nasa.gov/search/granule_values_by_collection.json?concept-id=C14758250-LPDAAAC_ECS

{
  "C14758250-LPDAAAC_ECS": {
    "cloud_cover": {
      "min": 0.0,
      "max": 100.0
    },
    // ... more fields
    "additional_attributes": {
      "ASTERProcessingCenter": ["GDS"],
      "ASTERGRANULEID": ["ASTL1A 0003042034040203230301", "ASTL1A 0003042034130203230302", "ASTL1A 0003051339230203230303", "ASTL1A 0003051339320203230304", "ASTL1A 0003060030050212200097", "ASTL1A 0003060030130212200098", "ASTL1A 0003060030220212200099", "ASTL1A 0003060030310212200100", "ASTL1A 0003060030400212200101", "ASTL1A 0003060030490212200102"],
      "ASTERMapProjection": ["N/A"],
      // ... more additional attributes
      "Band10_Available": ["Yes, band is acquired", "No, band was not acquired"],
      "Band11_Available": ["Yes, band is acquired", "No, band was not acquired"]
    }
  }
}
```

Estimate

Assumes implementation using elasticsearch aggregations with caching.

Tasks

- Determine fields to aggregate. (2)
 - Assume additional attributes, cloud cover, campaign, day night flag, orbital parameters, platforms, instruments, and sensors.
- Test elasticsearch query impact on performance (6)
 - Create sample elasticsearch query to aggregate all values
 - Run workload test
 - Execute query and monitor performance impact.
- Design (2)
 - Any additional design to do at the code level
- Initial Implementation (10)
 - First implementation won't include caching.
 - API
 - Query for aggregate data in elastic and extract response
- Stored Cache capability (10)
 - We don't have the ability to cache data that would be shared across instances. We should add this in a reusable way across the

- CMR. It could be a shared service or a library.
 - This would probably be implemented as a simple key value store mapped to Oracle or Elasticsearch.
- Updated Implementation to use caching (6)
 - Background job to fetch and store aggregates
 - Update implementation to fetch the data from the cache
- Integration test (6)

2+6+2+10+10+6+6=42